FONDAZIONE BRUNO KESSLER

CENTER FOR SENSORS & DEVICES

Interactive Data Visualization: Dash and Plotly Dashboards

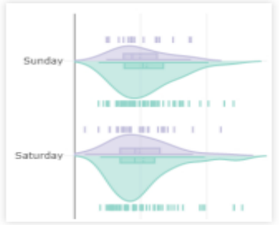
Christian Stefani
Center for Sensors & Devices
Fondazione Bruno Kessler

cstefani@fbk.eu
www.fbk.eu

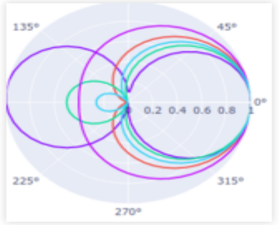
Plotly

Fundamentals

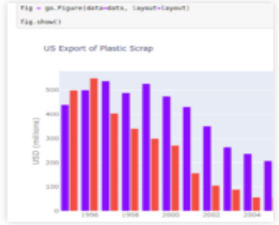
More Fundamentals »



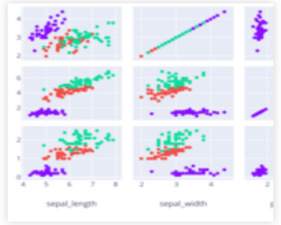
The Figure Data Structure



Creating and Updating Figures



Displaying Figures

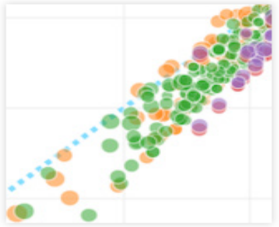


Plotly Express

Analytical Apps with Dash

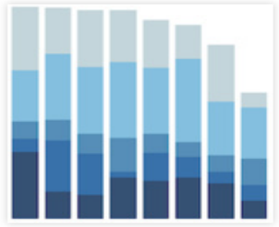
Basic Charts

More Basic Charts »

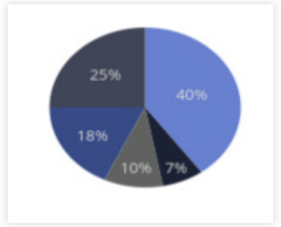


Scatter Plots

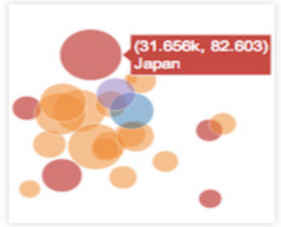
Line Charts



Bar Charts



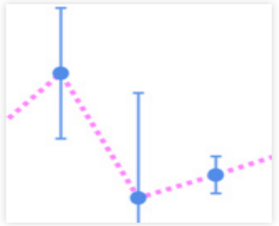
Pie Charts



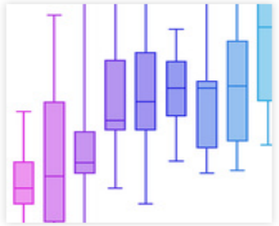
Bubble Charts

Statistical Charts

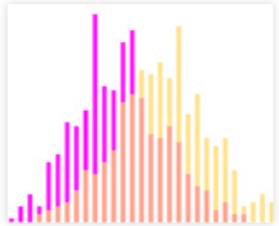
More Statistical Charts »



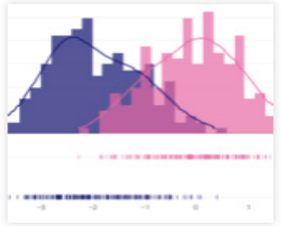
Error Bars



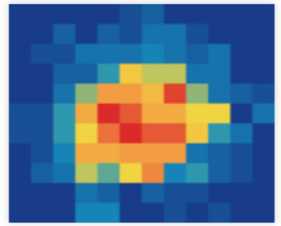
Box Plots



Histograms



Distplots



2D Histograms

Dash

All Apps (104)

Search applications...

Real-Time Object Detection

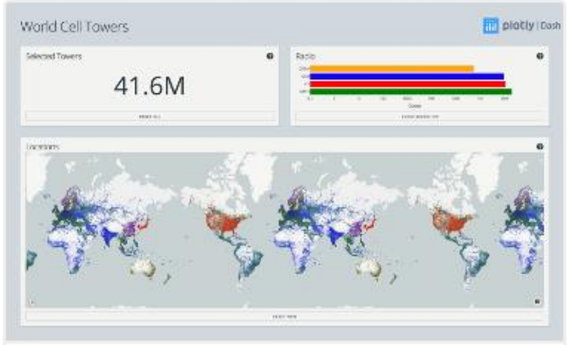
Image Processing | AI | ML | Computer Vision | (+1)

Manufacturing SPC Dashboard

Streaming | DAQ | Manufacturing

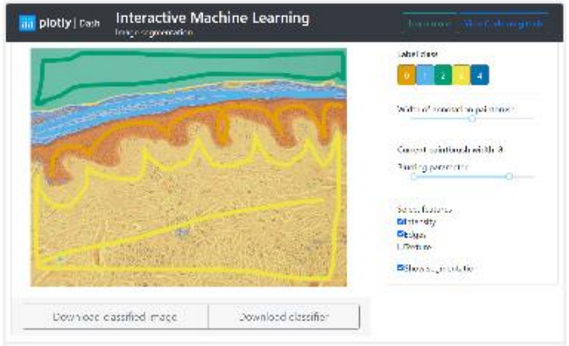
Wind Speed Dashboard for ...

Streaming | SQL | Energy



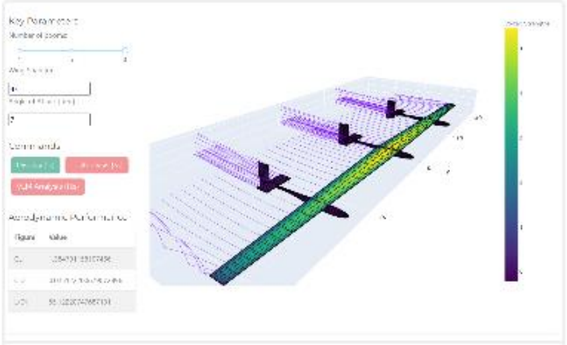
World Cell Towers

Geospatial | Datashader | Dask | Telecom



Interactive Image Segmenta...

ML | Healthcare | AI | Image Processing | (+1)



Aircraft CFD

Aerospace | CFD

Why Plotly?

- Fast and easy to implement simple plots
- Low code/low effort options using plotly.express
- (If desired) Extremely customizable
- Interactive plots by default

Why Dash?

- Available in R and Python
- Web-based
- Open source
- Easily versioned
- Fast, light-weight (built on top of Flask)
- Customizable with Flask
- Continuous integration/deployment

Plotly

Plotly graphs can be created:

1. With plotly.express for simple, quick plots (px)
2. With plotly.graph_objects (go) for more customization
3. With plotly.figure_factory for specic, advanced figures

We will spend most of our time on 1 and 2!

The importance of documentation

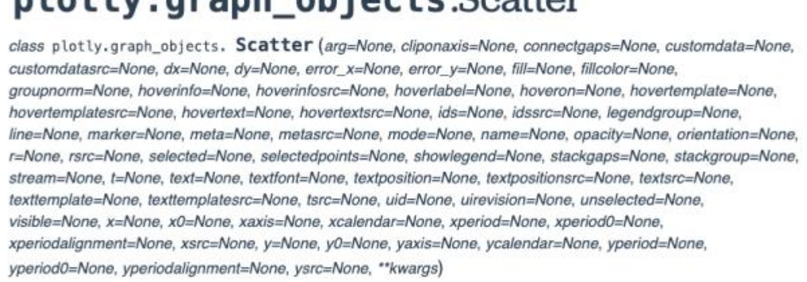
Save the links to key documentation!

1. Interactive, introductory docs (with many examples!)

https://plotly.com/python

2. Graph_objects pages for specic plots
    - Index here
    - For example, go.scatter here
3. The base go.Figure documentation linked here
    - Important when we cover update_layout() later!

The go.scatter documentation page:



plotly.graph_objects.Scatter

class plotly.graph_objects. **Scatter** (*arg=None, cliponaxis=None, connectgaps=None, customdata=None, customdatasrc=None, dx=None, dy=None, error_x=None, error_y=None, fill=None, fillcolor=None, groupnorm=None, hoverinfo=None, hoverinfosrc=None, hoverlabel=None, hoveron=None, hovertemplate=None, hovertemplatesrc=None, hovertext=None, hovertextsrc=None, ids=None, idssrc=None, legendgroup=None, line=None, marker=None, meta=None, metasrc=None, mode=None, name=None, opacity=None, orientation=None, r=None, rsrc=None, selected=None, selectedpoints=None, showlegend=None, stackgaps=None, stackgroup=None, stream=None, t=None, text=None, textfont=None, textposition=None, textpositionsrc=None, textsrc=None, texttemplate=None, texttemplatesrc=None, tsrc=None, uid=None, uirevision=None, unselected=None, visible=None, x=None, x0=None, xaxis=None, xcalendar=None, xperiod=None, xperiod0=None, xperiodalignment=None, xsrc=None, y=None, y0=None, yaxis=None, ycalendar=None, yperiod=None, yperiod0=None, yperiodalignment=None, ysrc=None, **kwargs*)

The Plotly Figure

A Plotly Figure has 3 main components:

- **layout**: Dictionary controlling style of the figure
  - One **layout** per figure

- **data**: List of dictionaries setting graph type and data itself
  - Data + type = a **trace**. There are over 40 types!
  - Can have multiple traces per graph

Inside the Plotly Figure

Let’s see inside an example Plotly **figure** object:

```
print(fig)
```

```
Figure({'data': [{'type': 'bar',
        'x': [Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday],
         'y': [28, 27, 25, 31, 32, 35, 36]}],
    'layout': {'template': '...',
               'title': {'font': {'color': 'red', 'size': 15},
                'text': 'Temperatures of the week', 'x': 0.5}}})
```

What do you think this graph will look like?

Inside out figure

```
Figure({ 'data': [{'type': 'bar',
     'x': [Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday],
     'y': [28, 27, 25, 31, 32, 35, 36]}],
     'layout': {'template': '...','title': {'font': {'color': 'red', 'size': 15},
     'text': 'Temperatures of the week', 'x': 0.5}}})
```

- Type ’bar’
- An X and Y axis with data noted
- A title with some text around temperatures of the week

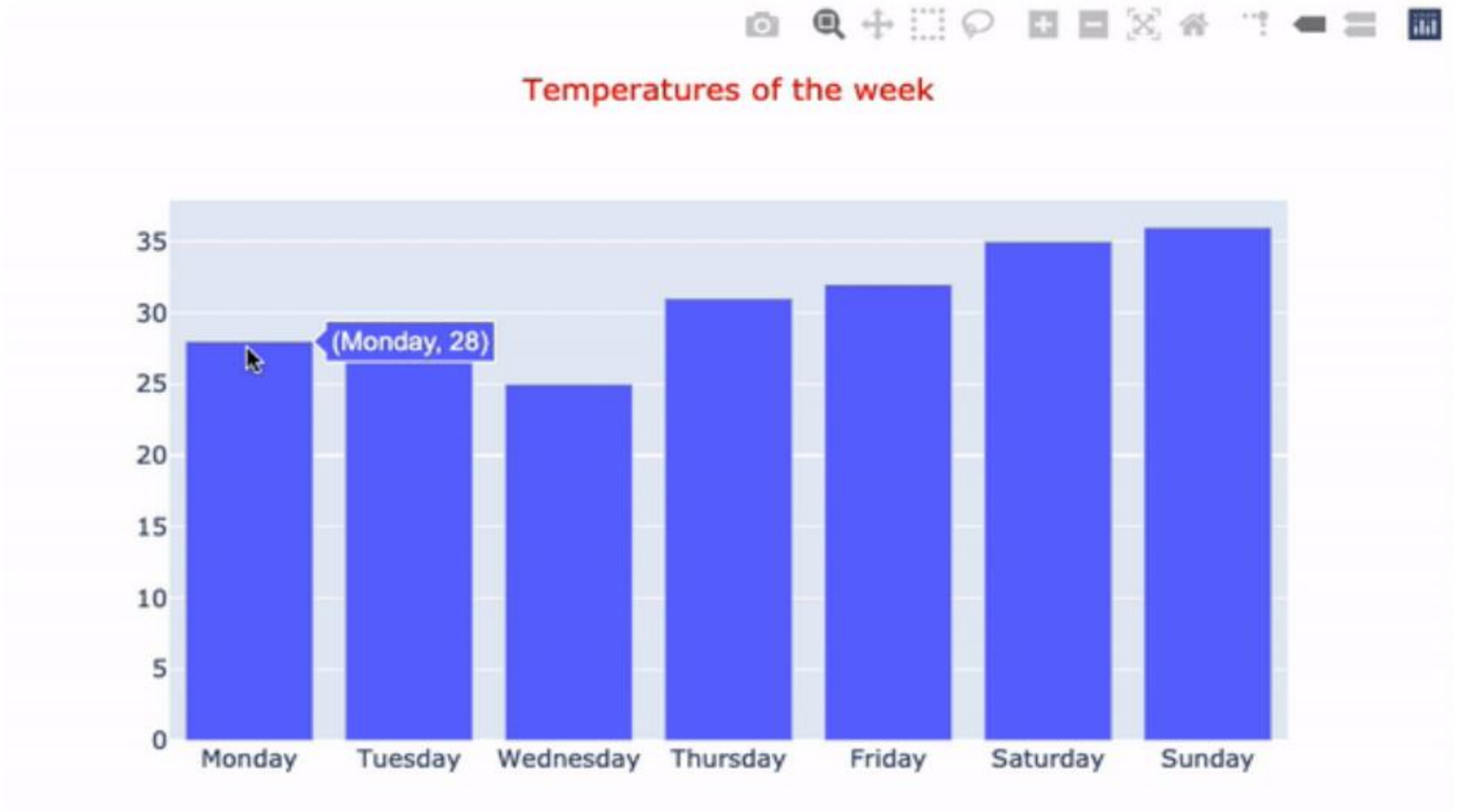
Guess: A bar chart of temperatures of the days of the week

Creating our Figure

```
import plotly.graph_objects as go
figure_config = dict({ "data": [{"type": "bar",
                "x": ["Monday", "Tuesday", "Wednesday",
                "Thursday", "Friday", "Saturday", "Sunday"],
                "y": [28, 27, 25, 31, 32, 35, 36]}],
              "layout": {"title": {"text": "Temperatures of the week",
              "x": 0.5, "font": {'color': 'red', 'size': 15}}}})
fig = go.Figure(figure_config)
fig.show()
```

Our Figure relevated

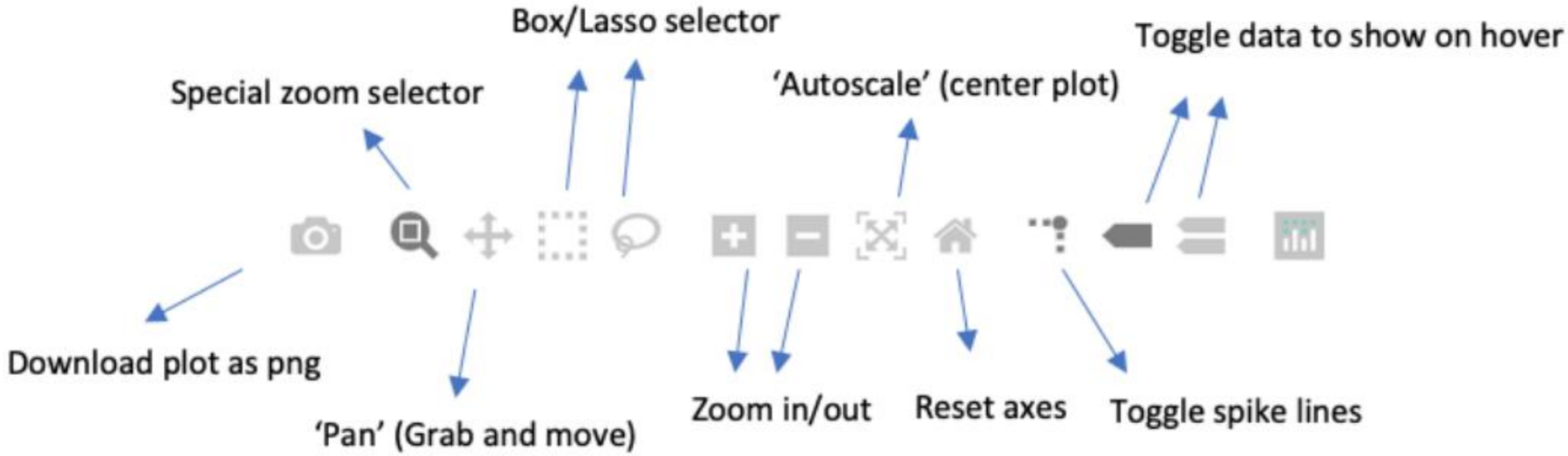
Let’s see what is produced!



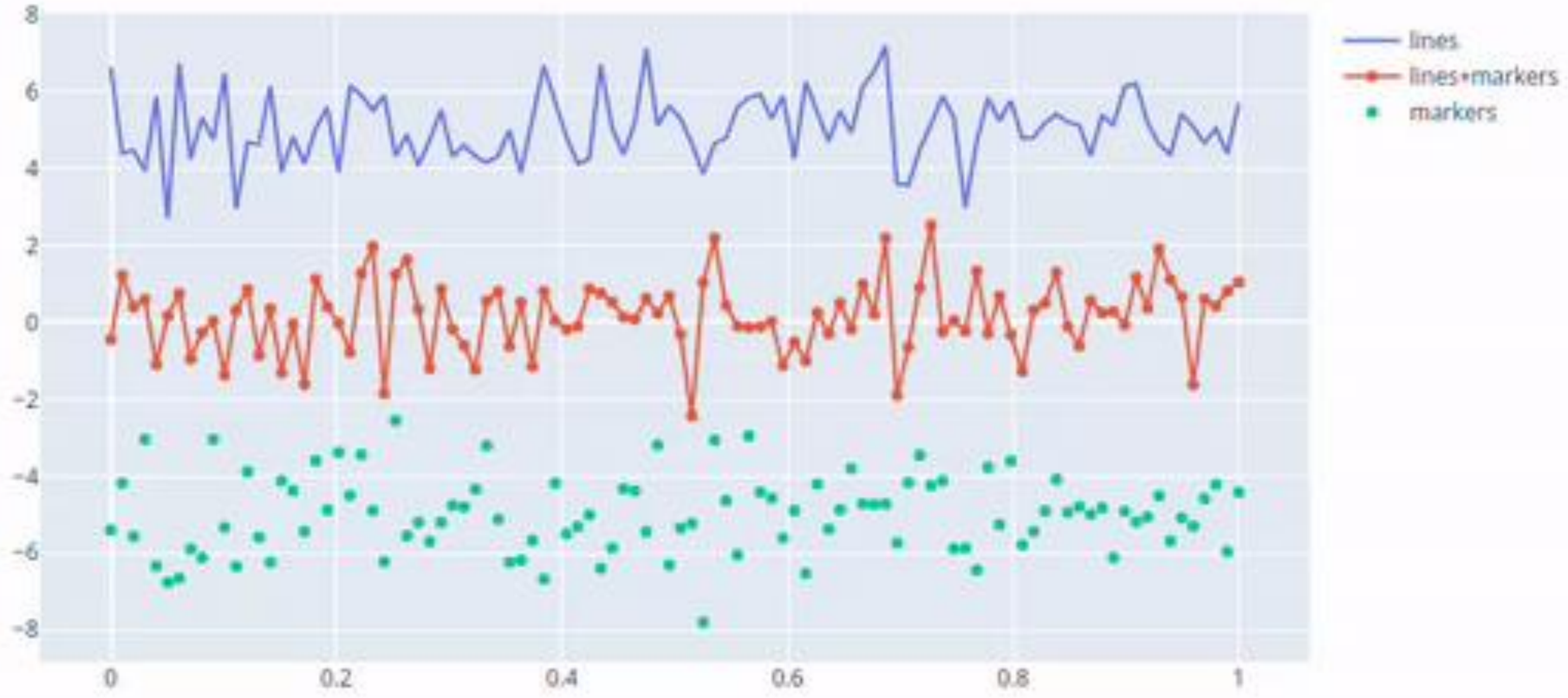
Plotly's instant interactivity

Plotly provides instant interactivity:

- Hover over data points
- Extra interactive buttons



Plotly’s instant interactivity



Univariate visualizations

Plotly shorcut methods:

1. `plotly.express`
   - Specify a DataFrame and its columns as arguments
   - Quick, nice but less customization

2. `graph_objects` `go`.x methods (go.Bar(), go.Scatter()) etc.
   - Many more customization options, but more code needed

What are univariate plots?

Univariate plots display only one variable

For analyzing the distribution of that variable

Common univariate plots:

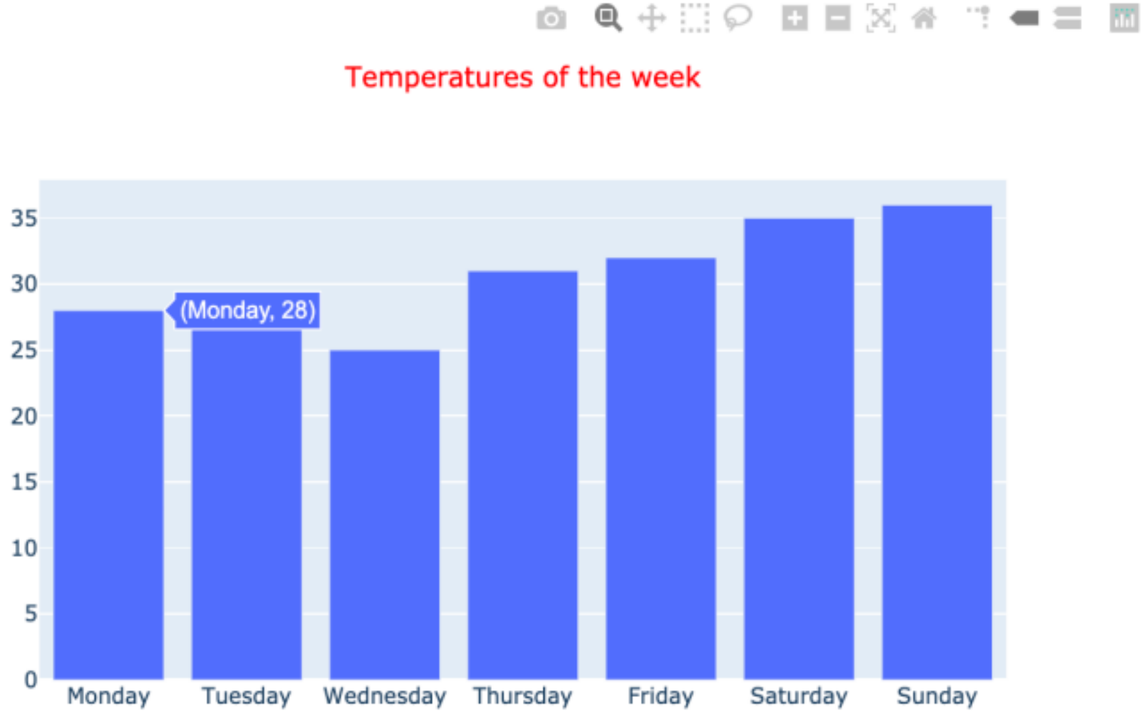
- Bar chart
- Histogram
- Box plot
- Density plots

Bar Charts

Bar charts have:

- X-axis with a bar per group
  - One group = one bar! (Hence UNIvariate)

- The y-axis height represents the value of some variable



Bar chart

Let's rebuild with **plotly.express**

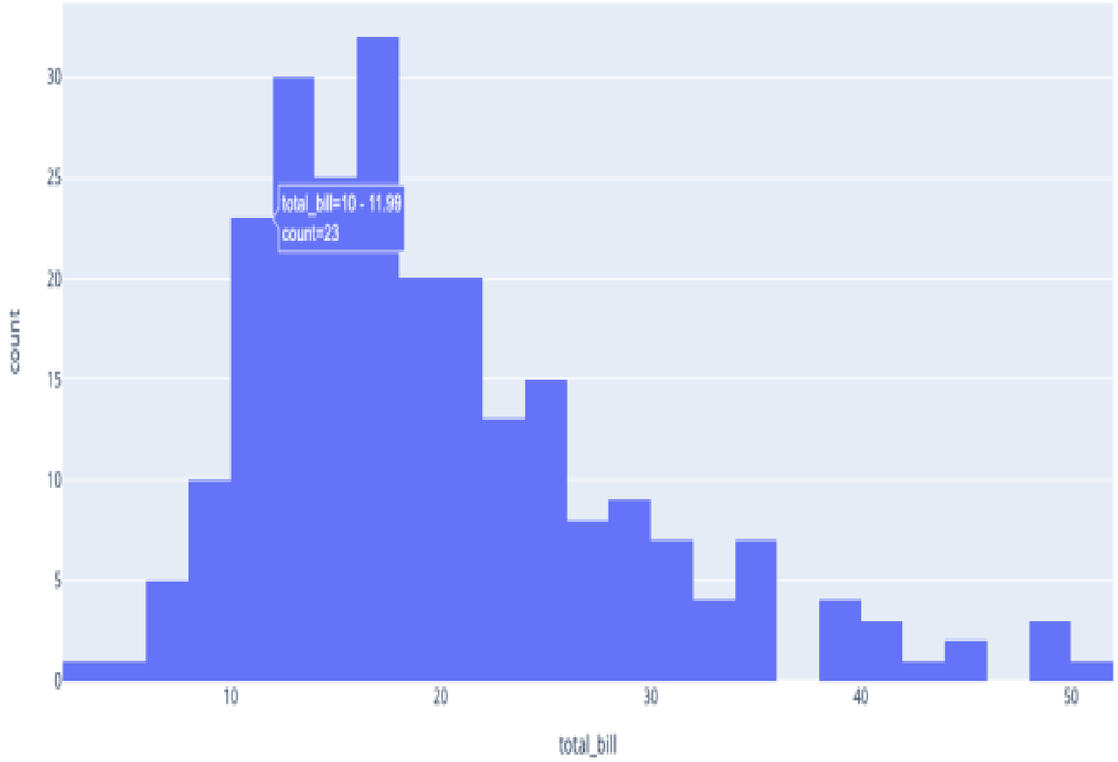
```
import plotly.express as px
weekly_temps = pd.DataFrame({
    'day': ['Monday', 'Tuesday',
            'Wednesday', 'Thursday', 'Friday',
            'Saturday', 'Sunday'],
    'temp': [28, 27, 25, 31, 32, 35, 36]})
fig = px.bar(data_frame=weekly_temps, x='day', y='temp')
fig.show()
```

Histograms

Histograms have:

- Multiple columns (called 'bins') representing a range of values
  - The height of each bar = count of samples within that bin range

- The number of bins can be manual or automatic



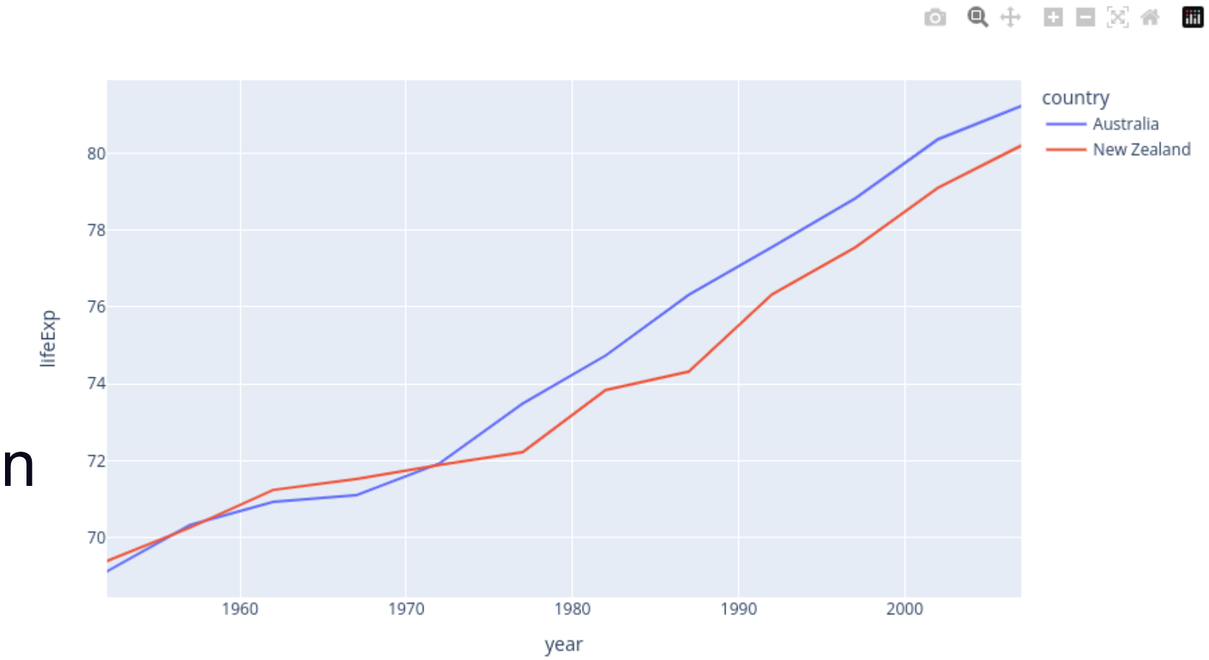
Line chart

```
import plotly.express as px

df = px.data.gapminder().query("continent=='Oceania'")
fig = px.line(df, x="year", y="lifeExp", color='country')
fig.show()
```

Line chart

Line chart has:

- Multiple lines(called trace) representing the values for each group
- Including the attribute ‘color’ we can divide the y values into different group



Customizing color

How to customize plots:

1. At gure creation if an argument exists (like color !)
2. Using an important function update_layout()
   - Takes a dictionary argument
   - E.g.: fig.update_layout({'title':{'text':'A New Title'}})

The method chosen depends on plot type how it was created.

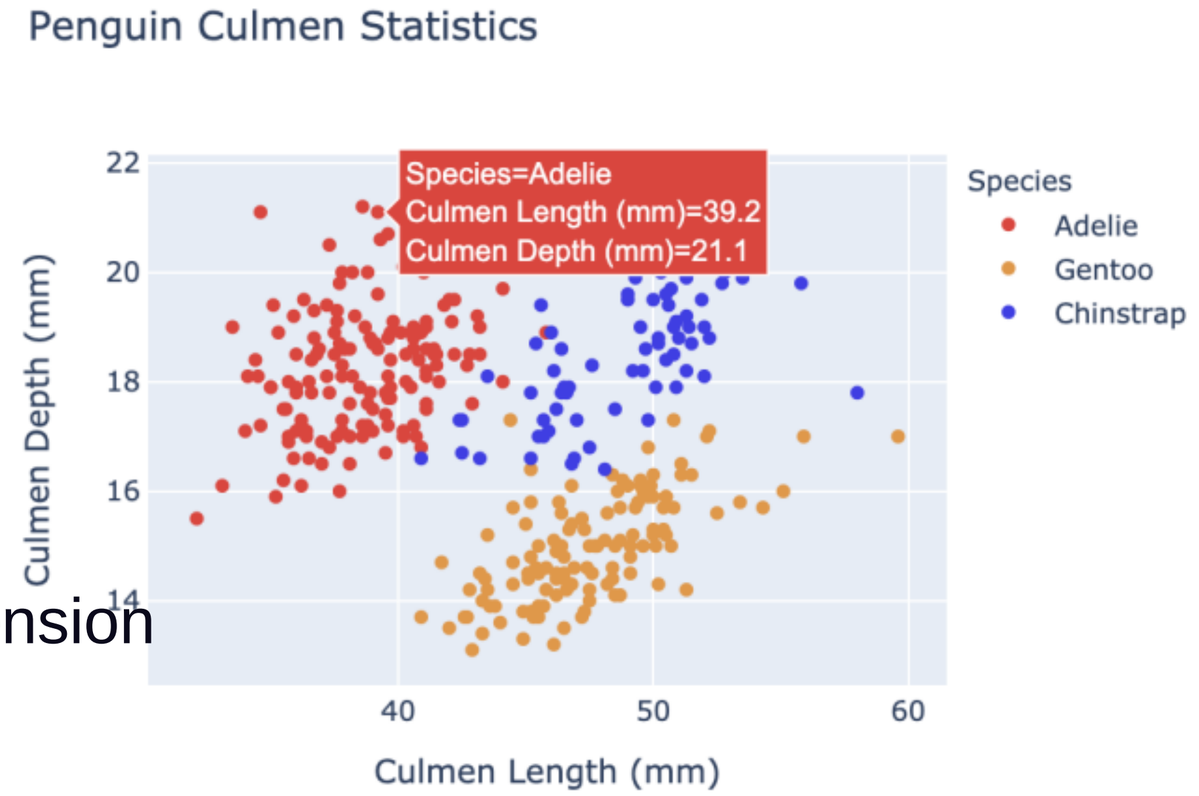
MANY properties possible — See the documentation

Why customize color?

Customizing color can help you

1. Make plots look awesome!

2. Convey analytical insights
   - Color in this scaerplot adds a 3rd dimension



Specifying colors in plotly.express

In plotly.express :

- Often a color argument (DataFrame column)
  - A dierent (automatic) color given to each category in this column
  - A color scale/range is used if numerical column specied

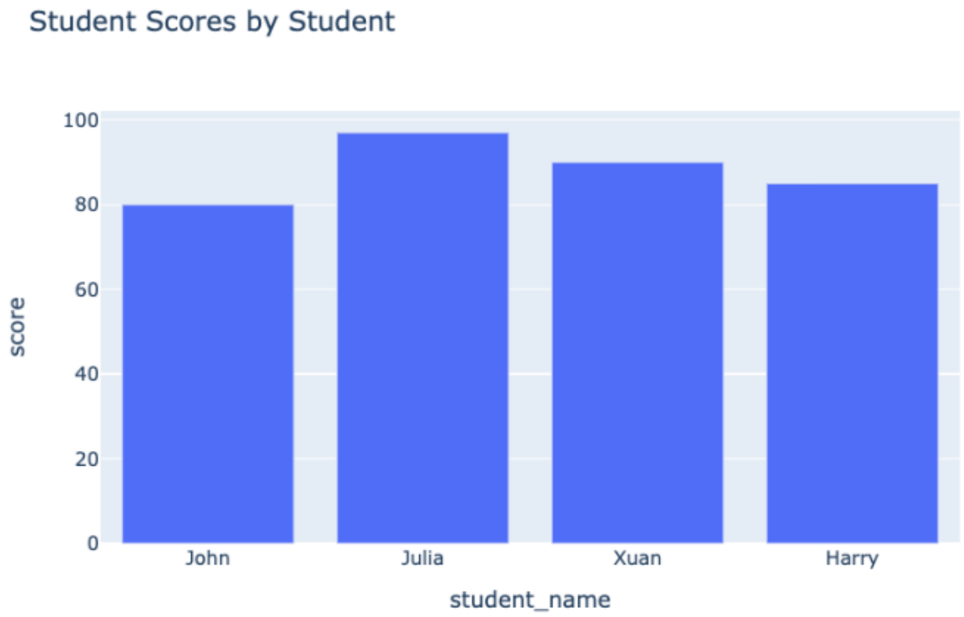
Our simple bar chart from a previous lesson
(adding a City column)

```
fig = px.bar(data_frame=student_scores,
       x='student_name',
       y='score',
       title='Student Scores by Student'
       color='city')
fig.show()
```

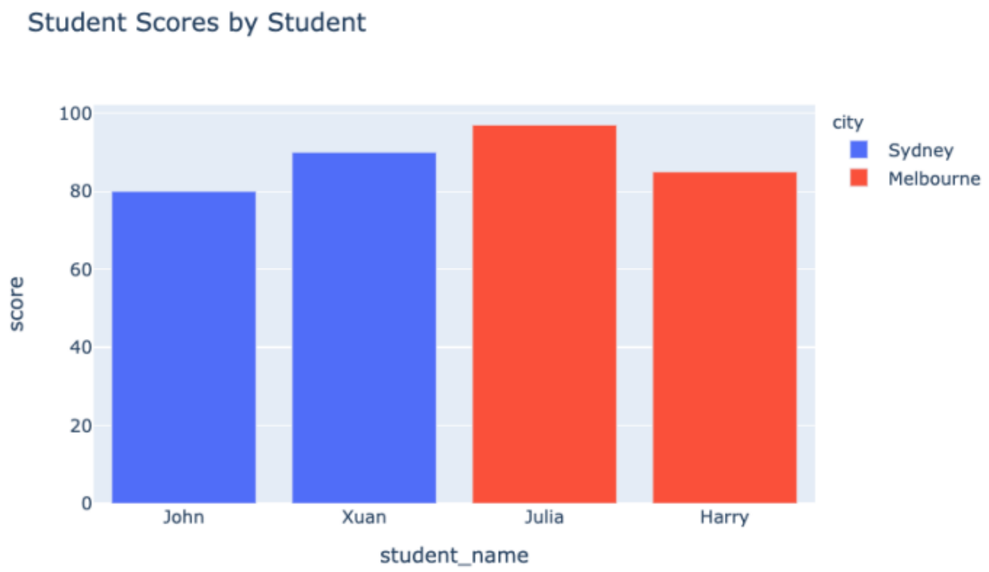
Make sure to check the documentation for each figure.

Our colors revealed

The plot before:



Our plot after:



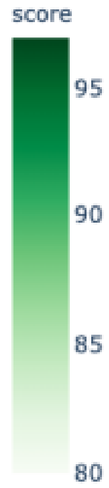
Specific colors in plotly.express

What if we don't like the automatic colors?

- color_discrete_map : A dictionary mapping specic categorical values to colors using a string RGB code specication — 'rgb(X,X,X)'

- Can also express (basic) colors as strings such as 'red', 'green' etc.

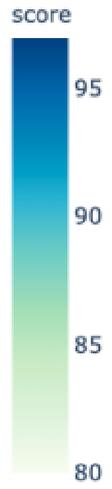
Color scales in plotly.express



You can create color scales too.

- Single color scales. For example, light to dark green.

- Multiple colors to merge into each other.
  For example, green into blue



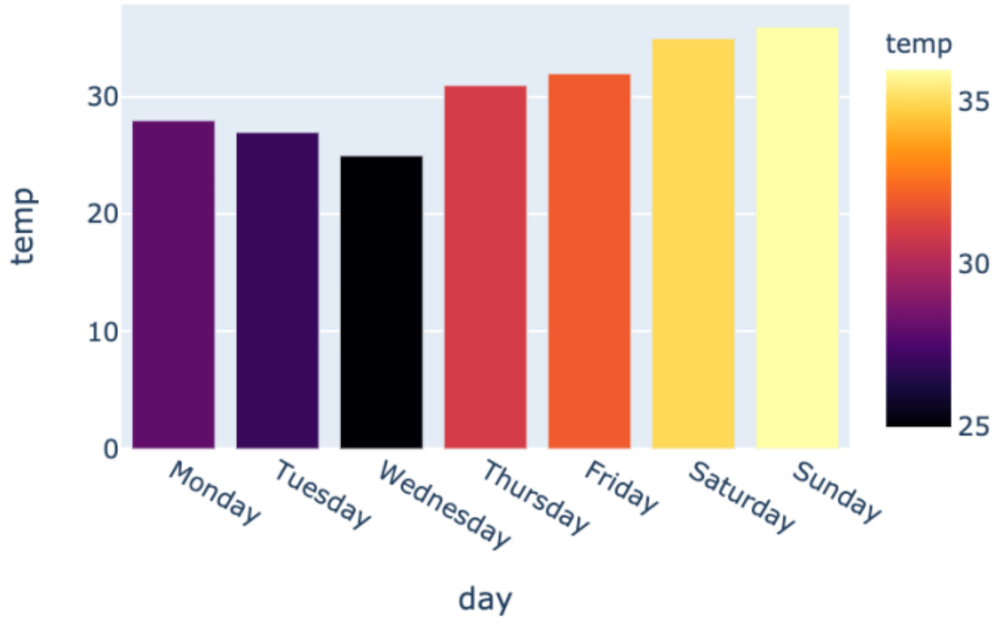
color_continuous_scale allows us to do this
with built-in or constructed color scales.

Using built-in color scales

Let's use a built-in color scale:

```
fig = px.bar(data_frame=weekly_temps,
    x='day', y='temp',
    color='temp',
    color_continuous_scale='inferno')
fig.show()
```

Our plot:



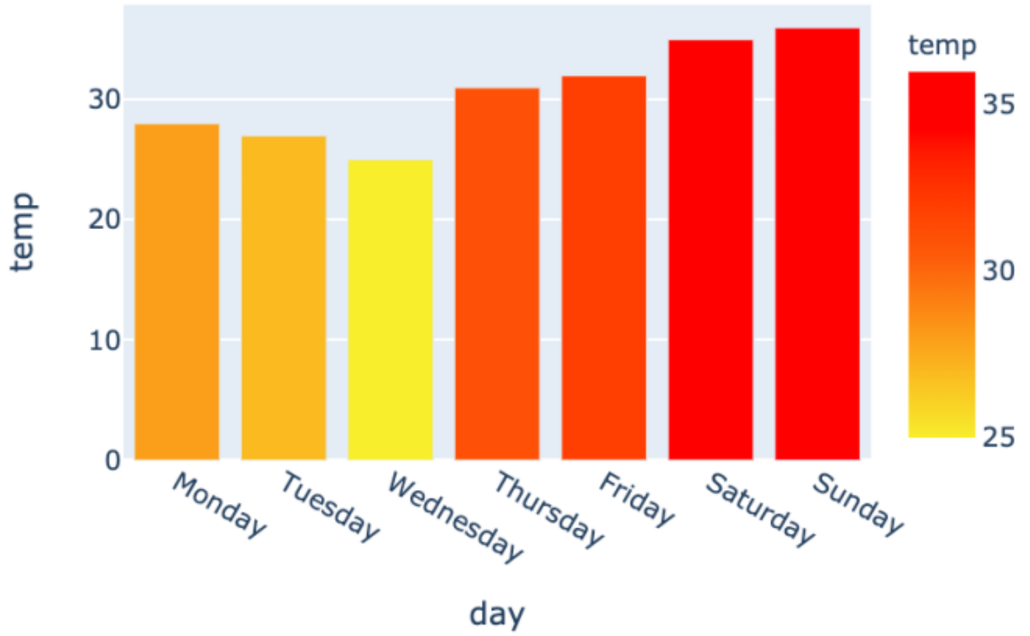
Many built-in scales available

Constructing our own color range

Let's construct our own color scale - yellow through orange to red

```
my_scale=[('rgb(242, 238, 10)'),
          ('rgb(242, 95, 10)'),
          ('rgb(255,0,0)')]
fig = px.bar(data_frame=weekly_temps,
    x='day', y='temp',
    color_continuous_scale=my_scale,
    color='temp')
```

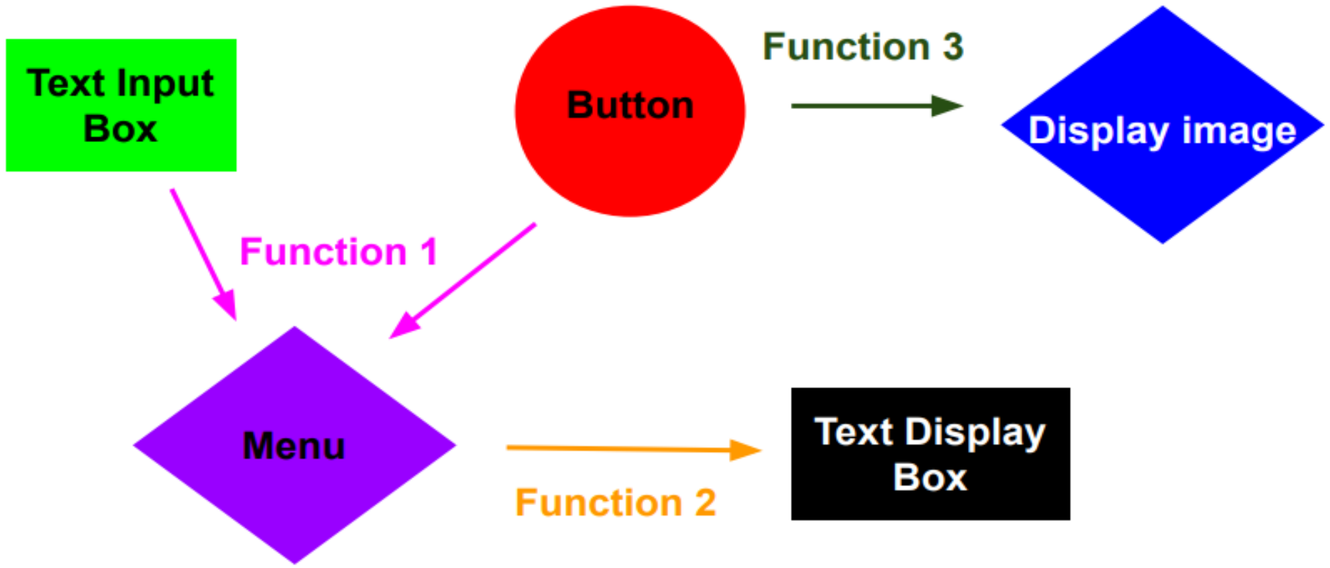
Our plot:



Introducing Dash

How does it work?

Essential Idea: Web page consists of a collection of components and collection of functions that are triggered whenever an aspect of a given
component changes.



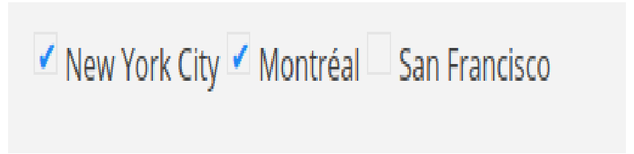
Core components

- dcc.Checklist
- dcc.ConfirmDialog
- dcc.ConfirmDialogProvider
- dcc.DatePickerRange
- dcc.DatePickerSingle
- dcc.Dropdown
- dcc.Graph
- dcc.Input
- dcc.Interval
- dcc.Link
- dcc.Loading
- dcc.Location
- dcc.LogoutButton
- dcc.Markdown
- dcc.RadioItems
- dcc.RangeSlider
- dcc.Slider
- dcc.Store
- dcc.Tab
- dcc.Tabs
- dcc.Textarea
- dcc.Upload

Populate webpage with various dash core components or html components.

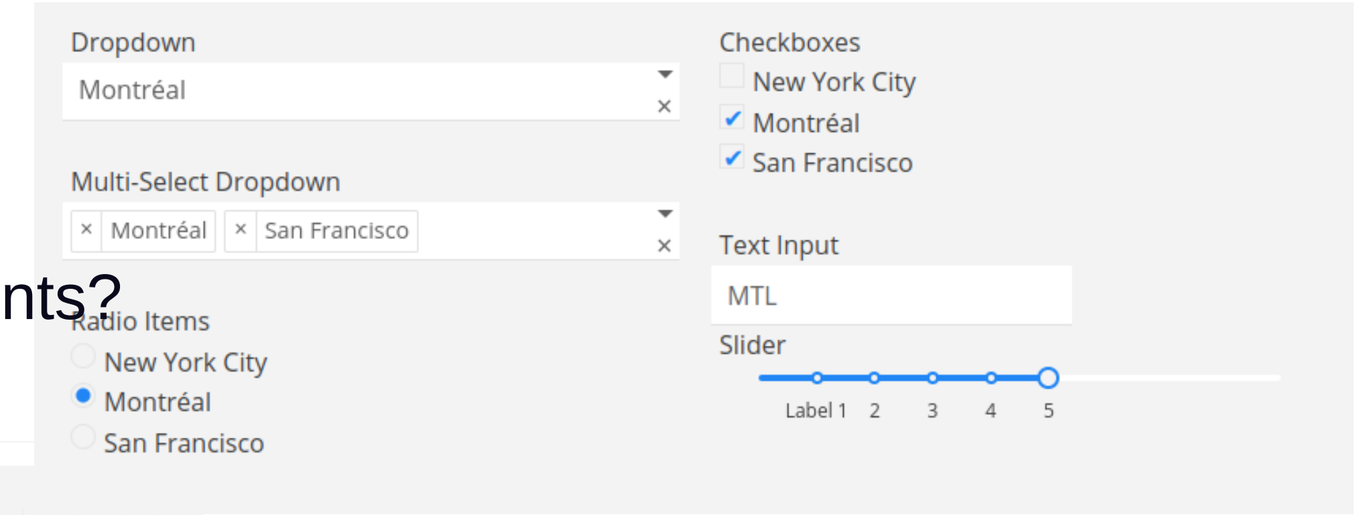
```
dcc.Dropdown(
    options=[
        {'label': 'New York City', 'value': 'New York City'},
        {'label': 'Montreal', 'value': 'Montreal'},
        {'label': 'San Francisco', 'value': 'San Francisco'},
    ],
    value='Montreal'
)
```

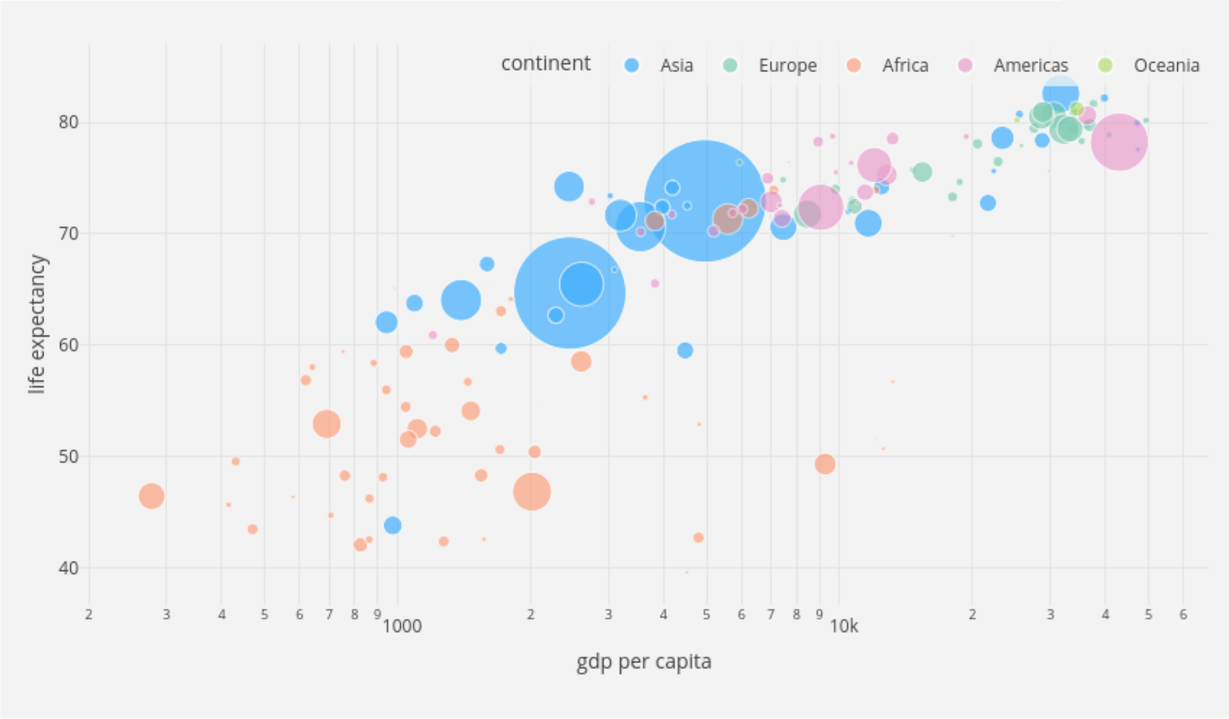

```
dcc.Checklist(
    options=['New York City', 'Montreal', 'San Francisco'],
    value=['Montreal']
)
```



Layout

But, how can we connect all components?



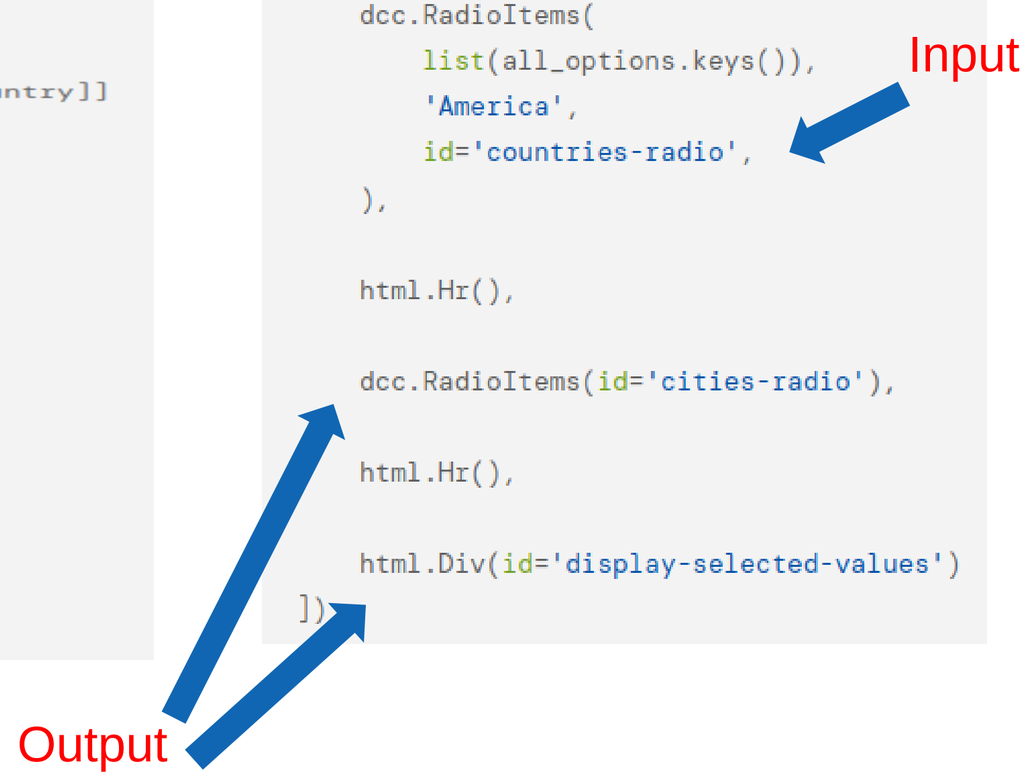


Callbacks

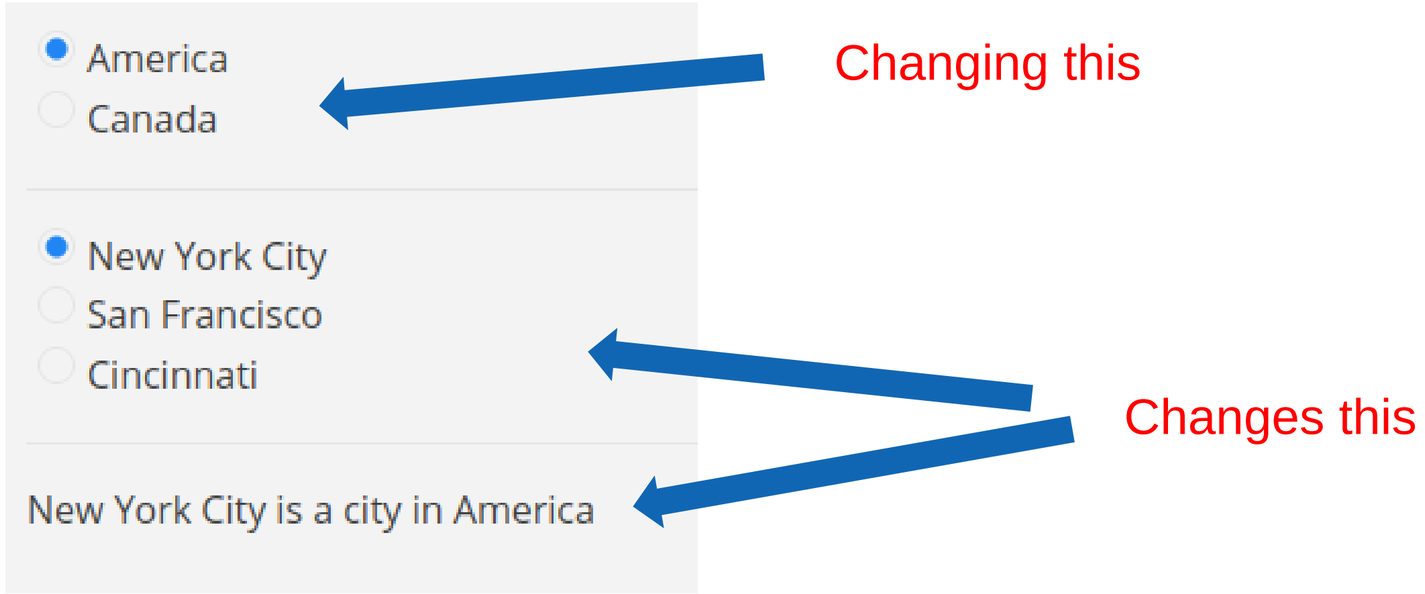
Define functions called callbacks that are triggered when an aspect of a given component changes

```python
@callback(
    Output('cities-radio', 'options'),
    Input('countries-radio', 'value'))
def set_cities_options(selected_country):
    return [{'label': i, 'value': i} for i in all_options[selected_country]]


@callback(
    Output('cities-radio', 'value'),
    Input('cities-radio', 'options'))
def set_cities_value(available_options):
    return available_options[0]['value']


@callback(
    Output('display-selected-values', 'children'),
    Input('countries-radio', 'value'),
    Input('cities-radio', 'value'))
def set_display_children(selected_country, selected_city):
    return u'{} is a city in {}'.format(
        selected_city, selected_country,
    )
```

```python
app.layout = html.Div([
    dcc.RadioItems(
        list(all_options.keys()),
        'America',
        id='countries-radio',
    ),

    html.Hr(),

    dcc.RadioItems(id='cities-radio'),

    html.Hr(),

    html.Div(id='display-selected-values')
])
```



Callbacks


America
Canada
New York City
San Francisco
Cincinnati
New York City is a city in America
Changing this
Changes this

Callbacks

- Everytime you change an input that is used in a callback, that callback will fire.
- Can define callbacks with multiple inputs.
    - Changing any one of those inputs will trigger the callback.
- Can use States instead, and a button to trigger the callback.

```
@app.callback(
    Output('display-diagnostic-test', 'children'),
    [Input('diagnostic-test-button', 'n_clicks')],
    [State('panel-number', 'value'),
     State('version', 'value'),
     State('batch', 'value'),
     State('pb-0', 'value'),
     State('pb-1', 'value'),
     State('pb-2', 'value'),
     State('pb-3', 'value'),
     State('pb-4', 'value'),
     State('pb-5', 'value'),
     State('pb-6', 'value'),
     State('pb-7', 'value'),
     State('pb-8', 'value'),
     State('pb-9', 'value'),
     State('serial-0', 'value'),
     State('serial-1', 'value'),
     State('serial-2', 'value'),
     State('serial-3', 'value'),
     State('serial-4', 'value'),
     State('serial-5', 'value'),
     State('serial-6', 'value'),
     State('serial-7', 'value'),
     State('serial-8', 'value'),
     State('serial-9', 'value'),
     State('debug', 'value')]
)
```

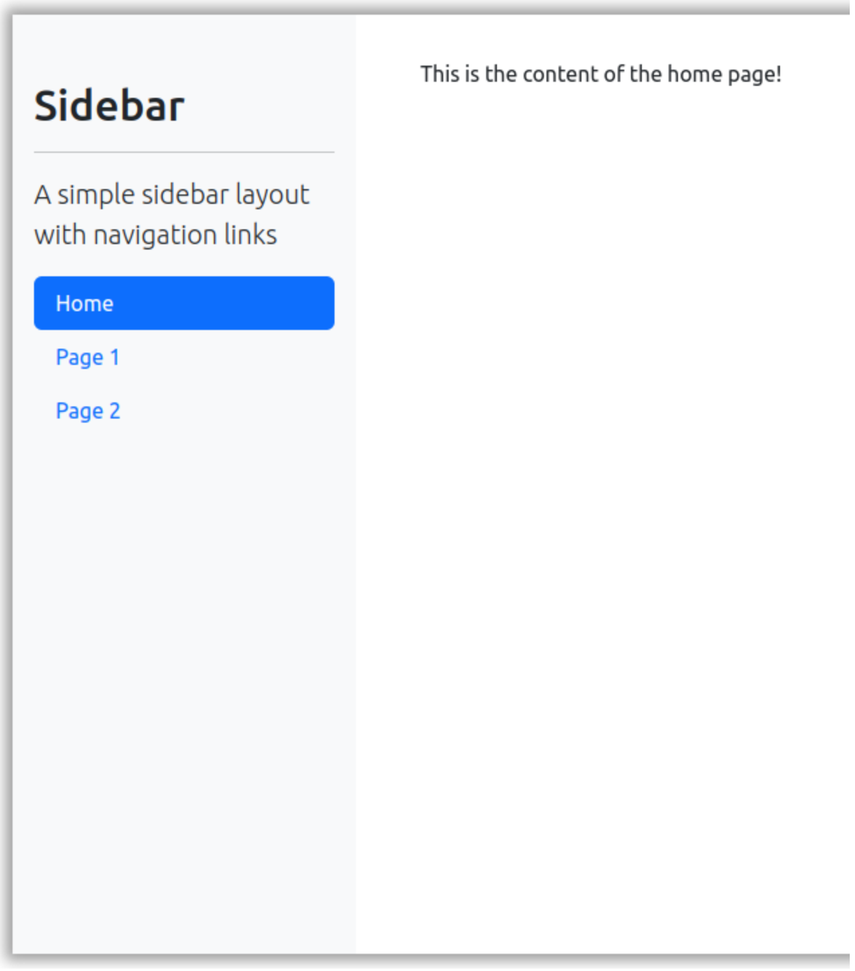
Css and dash-bootstrap-components

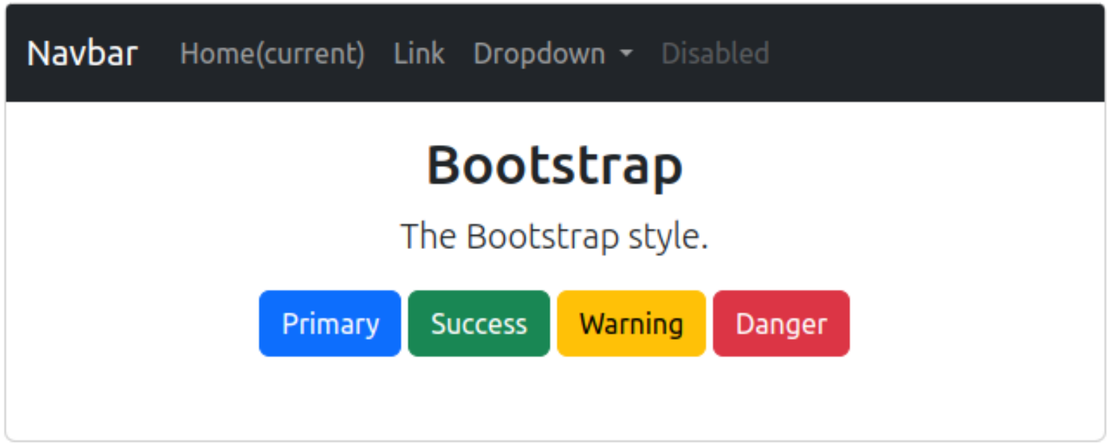
- Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in HTML.
- CSS describes how elements should be rendered on screen, on paper, in speech, or on other media.

```
app = dash.Dash(external_stylesheets=[dbc.themes.BOOTSTRAP])
```

- Bootstrap is the most popular CSS Framework for developing responsive and mobile-first websites.
- In particuraly, dash-bootstrap-components is a library of Bootstrap components for Plotly Dash, that makes it easier to build consistently styled apps with complex, responsive layouts.

Css and dash-bootstrap-components





Let's practice!

Installation

- pip install dash
- pip install dash-html-components
- pip install dash-core-components
- pip install dash-bootstrap-components
- pip install plotly
- pip install pandas

Dashboard

Avocado Analytics

Analyze the behavior of avocado prices and the number of avocados sold in the US between 2015 and 2018

Region

Albany

Type

Organic

Date Range

01/04/2015 → 03/25/2018

Average Price of Avocados

Avocados Sold

First steps

```
import pandas as pd
from dash import Dash, Input, Output, dcc, html
import plotly.graph_objects as go
```

```
data = (
    pd.read_csv("avocado.csv")
    .assign(Date=lambda data: pd.to_datetime(data["Date"], format="%Y-%m-%d"))
    .sort_values(by="Date")
)
```

```
app = Dash(__name__)
app.title = "Avocado Analytics: Understand Your Avocados!"


if __name__ == "__main__":
    app.run_server(debug=True)
```

Layout

- This is our container for the layout

```
app.layout = html.Div(children=[])
```

We are going to put all compentents into it

Dropdown, date picker, checkbox etc.

```
regions = data["region"].sort_values().unique()
avocado_types = data["type"].sort_values().unique()
```

First we have to save the values for the dropdown menus

Layout

First of all, we need to create the title with a little description.

```
html.Div(
    children=[
        html.H1(
            children="Avocado Analytics"
        ),
        html.P(
            children=(
                "Analyze the behavior of avocado prices and the number"
                " of avocados sold in the US between 2015 and 2018"
            ),
        ),
    ],
),
```

And then, the two dropdown menus.

```
html.Div(
    children=[
        html.Div(children="Region"),
        dcc.Dropdown(
            id="region-filter",
            options=[
                {"label": region, "value": region}
                for region in regions
            ],
            value="Albany",
            clearable=False,
        ),
    ]
),
```

```
html.Div(
    children=[
        html.Div(children="Type"),
        dcc.Dropdown(
            id="type-filter",
            options=[
                {
                    "label": avocado_type.title(),
                    "value": avocado_type,
                }
                for avocado_type in avocado_types
            ],
            value="organic",
            clearable=False,
            searchable=False,
        ),
    ],
),
```

Layout

```
html.Div(
    children=[
        html.Div(
            children="Date Range"
        ),
        dcc.DatePickerRange(
            id="date-range",
            min_date_allowed=data["Date"].min().date(),
            max_date_allowed=data["Date"].max().date(),
            start_date=data["Date"].min().date(),
            end_date=data["Date"].max().date(),
        ),
    ]
),
```

Set up our date picker (for a better date picker see Dash mantine)

Finally, we can add the two graph containers.

```
html.Div(
    children=[
        html.Div(
            children=dcc.Graph(
                id="price-chart",
                config={"displayModeBar": False},
            ),
        ),
        html.Div(
            children=dcc.Graph(
                id="volume-chart",
                config={"displayModeBar": False},
            ),
        ),
    ],
),
```

Callbacks

This is the head of our callback

- Which are the input elements?
- What we have to show?

```python
@app.callback(
    Output("price-chart", "figure"),
    Output("volume-chart", "figure"),
    Input("region-filter", "value"),
    Input("type-filter", "value"),
    Input("date-range", "start_date"),
    Input("date-range", "end_date"),
)
def update_charts(region, avocado_type, start_date, end_date):
    filtered_data = data.query(
        "region == @region and type == @avocado_type"
        " and Date >= @start_date and Date <= @end_date"
    )
```

Customize

And now try to customize again and again...